Formal Grammar and Human Factors Design of an Interactive Graphics System

PHYLLIS REISNER

Abstract—Formal grammatical description has not generally been applied in the human factors area, which traditionally draws on behavioral science for its methodology. This paper illustrates, by means of a detailed example, how formal grammatical description can be used as a predictive tool to compare alternative designs for ease of use and to identify design choices which could cause users to make mistakes.

The paper describes the human interface for two versions of an interactive graphics system intended for use by nonprogrammers. It presents the "action languages" for the two versions, then shows how these user languages can be described in terms of a production rule notation. Particular emphasis is given, in the notation, to actions the user has to learn and remember (i.e., to "cognitive" factors). The paper then presents predictions about human performance based on the formal description, and exploratory results of testing some of the predictions. Since the predictions are based on general properties of the formal description, the technique should also be applicable to other "action languages."

Index Terms—Action languages, analytic tools, comparison of design alternatives, ease-of-use measurement, ease-of-use prediction, formal description, human factors, man-machine interface, user errors, user-oriented design.

Introduction

In BOTH linguistics and computer science, formal grammars are used to describe languages precisely. In linguistics, a grammar is used to describe the rules for generating sentences in a natural language. These rules are usually assumed to be the ones a native speaker of the language follows implicitly in creating and in understanding sentences [2]. In computer science, a grammar is used to describe a programming language to a translator (e.g., compiler) or a compiler generator. The compiler can then be used to translate statements in the programming language into machine language [8], [10].

Formal grammatical description has not, however, been generally applied in the human factors area, which draws primarily on behavioral science for its methodology. Since human factors methodology rarely includes theoretical description or precise prediction based on formal models, the field is often considered "soft" or even ad hoc (at least, by some practitioners of other disciplines).

In this paper we illustrate, by means of a detailed example, how formal description can be used as a tool to aid the design of systems for ease of use. We are specifically concerned with "action languages" for interactive systems—the sequences of button presses, joystick motions, typing actions, etc.—performed by a user interacting with a terminal. The intent of the paper is to show that an action language can be formally described and that the formal description can be used to compare alternative designs for simplicity and consistency. We are particularly concerned, in the formal description, with "cognitive" factors—what a user has to learn and remember—rather than with physical actions themselves. By examining such a formal description rather than waiting for construction of a real, physical model, as required for behavioral testing, we hope to identify some design flaws early in the development cycle.

The particular system studied, ROBART, is an experimental, interactive color graphics system for creating slides for technical presentations. It is intended to be used by people without computer training. Two comparable versions of the system exist (ROBART 1 and ROBART 2). The two versions have essentially the same function but differ in the the design of the human interface.

The paper describes relevant portions of the two ROBART versions in some detail. It then presents a formal description of the ROBART 1 action language and makes predictions about human performance based on the formal description of ROBART 1 and matching parts of ROBART 2. Then, it presents results of exploratory tests of some of the predictions. The potential value of formal description of action languages is discussed, as well as other related work.

Description of ROBART

Background

ROBART 1 is an interactive program which runs on RAINBOW 1, an experimental color display system developed at the Research Division of IBM in San Jose, CA. ROBART 1 was originally developed by R. Williams to debug and demonstrate the RAINBOW 1 hardware [1]. It was designed from the "inside out," with primary emphasis being given to ease of constructing the hardware and to ease of programming, rather than "outside in," with primary focus on ease of use. The idea itself was imaginative, however, and the program has been used frequently to create slides for technical presentations and has also been used to create artistic pictures for journal covers [11]. A similar idea was independently conceived by R. Schoup at Xerox [14].

ROBART 2 is a program written for RAINBOW 2, a follow-on color display system. With experience gleaned from ROBART 1 as input, the human interface for ROBART 2 was carefully redesigned to improve ease of use. The redesign was the result of a collaborative effort between a psychologist

Manuscript received January 29, 1980; revised September 29, 1980.
The author is with the IBM Research Laboratory, San Jose, CA 95193.

0098-5589/81/0300-0229$00.75 © 1981 IEEE

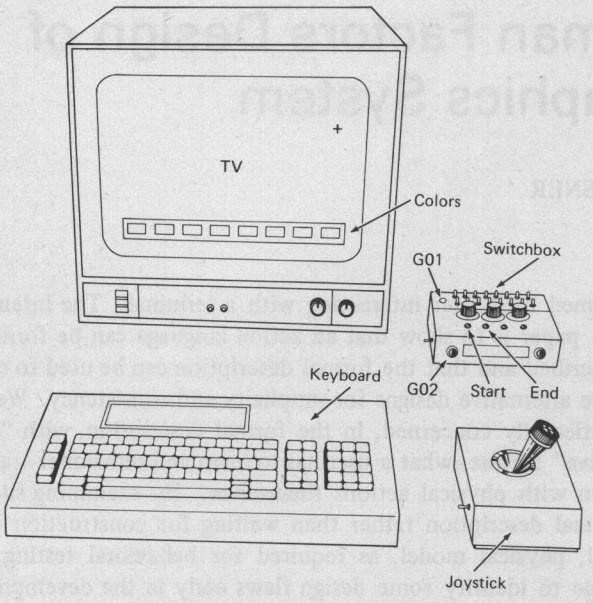

Fig. 1. ROBART 1 as seen by user.

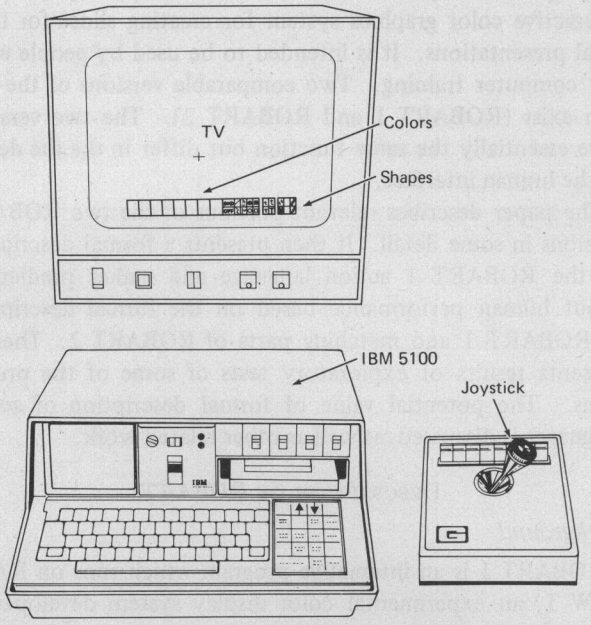

Fig. 2. ROBART 2 as seen by user.

and a computer scientist. The psychologist (the author) redesigned the user interface and the computer scientist, G. G. Langdon, redesigned the internal program structure [6]. The formal description was not used in the design of ROBART 2.

General Description

Both ROBART versions permit a user to create colored shapes–lines, rectangles, circles, etc. of any size and position and to type in color on a TV monitor. The two versions are illustrated in Figs. 1 and 2, and a picture created with ROBART 2, in Fig. 3.

ROBART 1 consists of a TV monitor with a color "paintbox" (menu of colors) displayed on it, a joystick for controlling the position of a cursor on the monitor, a switchbox

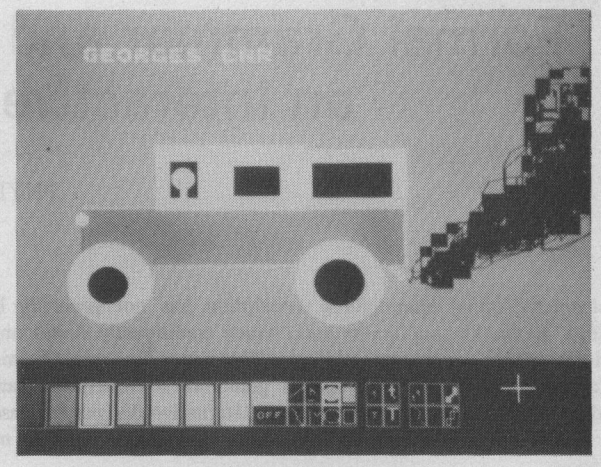

Fig. 3. Picture created by ROBART 2. Menu of colors and icons shown below.

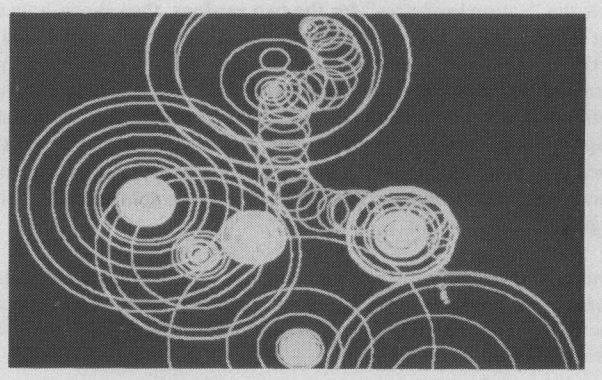

Fig. 4. Graphic art created with ROBART 1, using "continuous" circle function.

for selecting shapes and for other control functions, and a keyboard for entering text. ROBART 1 runs on an IBM System/7 computer and a RAMTEK display processor.

ROBART 2 also has a paintbox displayed on a TV monitor and a joystick for control of a cursor. It runs on an IBM 5100 computer, which provides the keyboard, and a Genisco display processor. The switchbox for selecting shapes in ROBART 1 has been replaced in ROBART 2 by a menu of icons (symbolic pictures) on the monitor. The paintbox and menu of icons can be seen in Fig. 3.

ROBART 1 Action Language

For brevity, only that portion of the description required for the ensuing discussion will be given. Readers wishing further detail can request an unpublished user's manual from the author.

Functions: The following shapes can be created with ROBART 1: line, rectangle, circle, horizontal and vertical lines, and "continuous" lines, rectangles, and circles. The continuous shapes are sequences of lines, rectangles, or circles which can be drawn free-form on the screen by moving the joystick. They are frequently used for artistic effects. The size and orientation of these continuous shapes can be varied. Fig. 4 is an example of the use of continuous circles for artistic effect, drawn by Musgrave [11, p. 253]. Two sizes of text exist and can be superimposed on any of the other shapes.

When text is superimposed on a colored background, it can be either superimposed directly or surrounded by a black area the size of the character box.

User Actions: Drawing a shape in ROBART 1 requires that the shape and color be selected, the size and location (and orientation if appropriate) be indicated, and any special parameters, such as text background, also indicated. Color and shape selections remain in effect until new selections are made.

Selecting Colors: Colors are selected by moving the cursor, controlled by the joystick, into the desired color in the paintbox area.

Selecting Shapes: Selecting a shape requires setting the appropriate switch or switches to the UP (on) position. For all but horizontal and vertical lines, it then requires pressing an interrupt button (GO 1) on the switchbox, to cause the switches to be read by the computer. The appropriate switches are

Shape	Switch Setting
line	LINE up
horizontal line	HORIZONTAL up
vertical line	VERTICAL up
rectangle	BOX up
circle	LINE up and BOX up
continuous line	CONTINUOUS up and LINE up
continuous rect.	CONTINUOUS up and BOX up
continuous circle	CONTINUOUS up, LINE up and BOX up

Defining Size, Location, and Orientation of Lines, Circles, and Rectangles: Two points on the screen suffice to define these shapes. The points are indicated by positioning the cursor at the first point, simultaneously pressing START and interrupt (GO 2) buttons on the switchbox, then positioning the cursor at the second point and simultaneously pressing END and GO 2. For lines, the start and end points are the two ends of the line; for circles, the center and any point on the circumference; and for rectangles, any two diagonal corners.

Connecting Lines: In order for the user to create a sequence of connected straight lines, the system has been designed so that the endpoint of one line is treated by the computer as the beginning of the next. The purpose of this design was to reduce the amount of human effort required. Thus, to connect lines the user merely has to draw the first line, then indicate the endpoints of successive ones. In order for this strategy to hold for horizontal and vertical lines, the user should not push the GO(1) button after the HORIZONTAL or VERTICAL switches have been set. (GO 1 resets an internal parameter, thus losing the position of the endpoint.)

Defining Size, Location, and Orientation of Continuous Shapes: Continuous shapes are drawn "continuously" as the joystick is moved. The size and direction are controlled by rotating a knob on top of the joystick. A switch (set to SIZE or ANGLE) determines whether size or direction is controlled by the knob. In order that a user will not have to move his hand from the joystick to start and stop his "drawing" with these shapes, turning the knob completely off (counterclockwise) prevents drawing from occurring.

ROBART 2 Action Language

The ROBART 2 action language is similar in many respects to that of ROBART 1. The method for selecting colors is the same. The method for defining size and location of lines, circles and rectangles is the same, with the exception that a single EXECUTE key on the keyboard replaces the START, END, and GO(2) buttons.

One major difference is the method of selecting shapes. The switchbox has been replaced by icons on the screen. These are selected in basically the same way as colors, by dipping the cursor into the appropriate icon. There is one icon for each shape, rather than the varying number of switches per shape in ROBART 1. The interrupt button (GO 1) is no longer required. A second difference between the two versions is the method of starting continuous shapes. In ROBART 2 it is necessary to position the cursor and press the EXECUTE key, thus requiring the user's hand to be lifted from the joystick, unlike the ROBART 1 design, which is more economical of motion. A third difference is the method for connecting lines. In ROBART 2 it is not possible to connect lines by simply indicating successive endpoints. Both the start and endpoints of all lines must be indicated.

The functions available are almost identical. ROBART 1 has continuous circles where ROBART 2 has continuous open rectangles. In the testing to be described later, we will treat these as a single comparable shape.

The effect of these design differences on human performance will be discussed in later sections. As with ROBART 1, we have not presented a complete discussion of the ROBART 2 action language. Readers desiring further information can request an (unpublished) user's manual from the author.

Formal Description

To describe the ROBART 1 action language, we will use a production rule grammar. This form of description is familiar to linguists and commonly associated with Chomsky [2]. It is also familiar to computer scientists, as Backus-Naur form, or BNF [12]. A production rule grammar describes a language as a set of *rules* for describing correct "strings" (e.g., sentences) in a language. Any particular string can then be described by the particular rules involved in producing it. The structure of the string can be shown by a tree diagram based on the rules.

Traditionally, a production rule grammar consists of:

1) a set of terminal symbols (the words in the language);

2) a set of nonterminal symbols (invented constructs used to show the structure of the language, e.g., "noun phrase,");

3) a starting symbol (e.g., "S", for sentence);

4) the metasymbols "+", "|", and "::=" (some common meanings for these are "and," "or," and "is composed of," respectively);

5) rules constructed from the above (e.g., S ::= noun phrase + verb phrase).

ROBART 1 Grammar

Terminal Symbols: The terminal symbols for a ROBART 1 grammar represent actions the user has to learn and remember, e.g.,

(PUT) LINE SWITCH UP
(ROTATE) KNOB PARTIAL CLOCKWISE
(POSITION) CURSOR AT CIRCUMFERENCE

We will call these terminal symbols "cognitive terminals" and the grammar thus described a "cognitive grammar." The word "cognitive," means "having to do with *knowing*" in a very broad sense, and thus includes understanding, learning, and remembering. We have chosen to write a "cognitive grammar" for ROBART because learning how to use a system and remembering how to use it after a delay will be of primary importance for the population we are considering: nonprogrammers doing nonroutine tasks. Other kinds of terminal symbols are possible and will be discussed later.

Terminal symbols are indicated in small capital letters. The word NULL means "no user action required." Words in parentheses are actions which, for brevity, are eliminated from the rules shown in the Appendix.

Nonterminal Symbols: The nonterminal symbols represent sets of similar actions that can be grouped together and described in the same way, e.g.,

(draw) colored shape
(draw) continuous shape

Nonterminals are indicated in lower case letters.

Starting Symbol: For both ROBART grammars, the starting symbol for the action language is

(DRAW) PICTURE

This represents a high-level task to be performed by the user. Thus, each picture to be created by the action language is analogous to a sentence in English.

Metasymbols: The metasymbols have been assigned the following meanings. The symbol "::=" means "is composed of." The symbol "|" means "or." The symbol "+" has been defined to mean concatenation, or "is followed by, in order." Thus, "a+b" means "a followed by b" *in that order*. To indicate that order is immaterial, which it sometimes is, we write a+b|b+a. We have also introduced a metasymbol, "-", to mean "simultaneous."

Rules: The rules of the grammar are given in the Appendix. In some cases, rules with only one nonterminal symbol on the right-hand side have been introduced, even though this unnecessarily expands the number of rules. This has been done so that the abstract structure common to both ROBART versions can later be shown.

How to Read The Notation

For those readers unfamiliar with BNF notation we present a brief description of how to read the ROBART rules.

Rule 1 gives a "recursive" definition, which says, essentially, that a picture consists of either a colored shape or an indefinite number of colored shapes. Since we are writing an "action grammar," and, in fact, a *cognitive* action grammar, we could read the above rule more precisely as "to know how to create a picture, the user has to know how to create either a colored shape or a series of colored shapes."

Rule 2 says that a colored shape consists of either a color followed by a shape or a shape followed by a color. Since the user can select either the shape first or the color first, both parts of the right-hand side are required.

Rule 4 says that a new color is selected by dipping the cursor into one of the colors. Small capital letters are used to indicate actual possible user actions. For brevity, we have not listed all the colors.

Rule 7 says that there are three kinds of shapes: discrete, continuous, and text.

Rule 9 says that to create a discrete shape the user has to select it and describe it. Since the order of the operations in this case must be as given, the terms are not permuted as in Rule 2, where either order was possible.

Other rules follow the same pattern.

EXPERIMENTATION

In the following section we will make predictions about ease of use of the two ROBART versions and give results of some exploratory tests of some of the predictions. We call the results "exploratory" since we had two existing systems to study and it was not possible to control all the factors we wished. We did, however, supplement the objective testing (measurements of time and accuracy) with probing questionnaires in an attempt to understand the truth of the matter. The testing was as follows.

Ten office workers from a temporary office agency learned both ROBART versions, primarily using the manuals and other tutorial materials available. (ROBART 2 had on-line tutorials for indicating size and location of shapes). The experimenter supplemented the self-teaching when absolutely required, being as consistent as possible for the two versions. This was done so that the principal test (a memory test) could be completed in the time available. A maximum of two hours per subject was allotted to the self-learning phase. To structure the learning process, subjects were given a list of simple tasks to perform (e.g., draw a green line). Half the subjects learned ROBART 1 first and half, ROBART 2.

Subjects were then given an "immediate memory" test. That is, the manuals were removed and the subjects were asked to demonstrate the tasks, using the same list. Testing time was roughly 30 to 40 min, on the average, for each group. A maximum time per item (2 min) was set to preclude excessive trial and error. Success or failure on each item was noted and the kinds of errors observed and classified. Although not of primary interest for testing the predictions, overall learning times were also noted.

In addition, the objective test was supplemented by structured questionnaires which asked for comparative judgments about specific features of each language, comparisons of features between the two languages, overall preferences, and reasons for the preferences.

USING THE FORMALISM TO COMPARE DESIGN ALTERNATIVES

Three aspects of the formalism are particularly useful for comparing design decisions: 1) the number of different terminal symbols, 2) the lengths of the terminal strings for particular tasks, and 3) the number of rules necessary to describe the structure of some set of terminal strings. The first repre-

sents the total number of different action steps in the language (the number of "words"). The second represents the number of steps the user has to perform for some given subtask (the "sentence lengths"). The third represents the consistency (or lack of it) in the steps required for a set of related subtasks (the number of different sentence types to say similar things). We will refer to the last two aspects as "string simplicity" and "structural consistency," respectively.

The design alternatives examined in the experiments involved these last two aspects. After presenting the results of the experimentation, we will discuss the relative importance of all three aspects.

String Simplicity

To illustrate "string simplicity" we will concentrate on the actions for "selecting" shapes.

Without regard to design, common sense would tell us that all shapes should be selected with equal ease in ROBART 1; that the same should be true of ROBART 2; and that it should be just as easy to select a shape in ROBART 1 as in ROBART 2.

Examination of the rules in the Appendix, however, shows us that this may not be the case. To select a line in ROBART 1, for example, we see from rule 17 that the user must "undo" (reset to the off position) any previously selected switches and then set the line switch. Ignoring the "undo" portion for the moment, we see from rule 19 that setting line involves putting the line switch up.

From rule 25, we see that both line and box switches must be set for circle. Continuing in the same way for other shapes, we see that the lengths of the terminal strings for setting shapes vary, with line and rectangle requiring one switch; circle, continuous line, and continuous rectangle requiring two, and continuous circle requiring three. For "selecting" shapes (rather than "setting switches"), the undo step and also the GO(1) step (rule 13) must be added. Notice that we distinguish between "selecting a shape," "selecting a switch," and "setting a switch." Selecting a shape includes selecting the appropriate switch or switches and pressing the GO(1) button, as in rule 13 in the Appendix, for example. Selecting a *switch* in turn, includes setting any previously chosen switches to the off position ("undo"-ing the unwanted switches), then setting the wanted switch to the on position. An example of this is rule 17 in the Appendix. Setting a switch is the final action of putting the switch in the "up" position, as in rule 19.

While the lengths of the terminal strings for selecting shapes varies in ROBART 1, in ROBART 2 selecting any of the above six shapes requires only one step, dipping the cursor into the appropriate icon, e.g.,

select line ::= CURSOR IN LINE ICON
select circle ::= CURSOR IN CIRCLE ICON
select continuous line ::= CURSOR IN C-LINE ICON

We would therefore expect the following.

Prediction 1: Learning and/or remembering how to select shapes in ROBART 1 should vary in difficulty.

If all else were equal, we would expect the shapes described by the shorter strings to be easier than the longer ones (e.g., line and rectangle easiest, continuous circle hardest).

We also expect the following.

TABLE I
NUMBER OF SUBJECTS (OF 10) UNABLE TO SELECT THE GIVEN SHAPE

	ROBART 1	ROBART 2
line	0	0
box	4	1
circle	8	0
continuous line	2	0
continuous box	6	0
continuous circle	9	1
(cont. open box in ROBART 2)		

Prediction 2: Learning and/or remembering how to select shapes in ROBART 2 should *not* vary in difficulty.

Prediction 3: Learning and/or remembering how to select any shape in ROBART 2 should be easier than selecting the corresponding ROBART 1 shape.

In Prediction 3 we are ignoring the fact that ROBART 1 uses switches and ROBART 2 uses icons. We were not able to control this difference, but tried to discover its importance in the subjective questionnaires.

Table I shows the number of subjects unable to select a given shape, for both ROBART versions. The data given in the table for ROBART 1 include the undo and GO(1) steps since we are comparing the entire sequence of steps required for "selecting" shapes. It is interesting to note that the major portion of the errors arose from the switch setting action. Data for the switch setting portion alone, omitting errors caused by forgetting the GO(1) button or the "undo" steps were, starting with "line" in the table, 0, 3, 8, 2, 5, and 8, respectively.

It can be seen from Table I that, as predicted, the ROBART 1 shapes differ in difficulty. This result is statistically significant (Cochran, $p < 0.05$) [15]. Averaging the results for shapes requiring one switch (line and rectangle), two switches, and three, we find the order being as expected, the averages being 2, 5.3, and 9, respectively. With a rank correlation test (Spearman), this is a perfect correlation.

The above test involved some uncontrolled factors (hardware, specific labels used, possible contamination from successive steps, icons versus switches, etc.). We were dealing with existing systems rather than with a "pure" laboratory experiment. We therefore supplemented the testing with detailed questionnaires and found that, when subjects were asked which version they thought was easier to learn and remember for each of the shapes, the results were similar. The reason given was the number of switches. Likewise, when subjects were asked to compare ROBART 1 shapes, similar results were obtained (the more switches, the harder).

Table II shows the subjective data comparing the ROBART 1 and ROBART 2 versions. Each shape was judged to be easier in ROBART 2 than the corresponding ROBART 1 shape. These results were, individually, statistically significant (sign test, each $p <= 0.05$). Four reasons were given for these preferences, with the number of switches to change being ranked the most important. (The others were, in descending order, remembering undo, remembering GO, and changing the visual field.)

TABLE II
NUMBER OF SUBJECTS (OF 10) JUDGING THE GIVEN SHAPE EASIER TO LEARN AND REMEMBER IN ROBART 1 OR ROBART 2 ("NO DIFFERENCE" RESPONSES NOT INCLUDED)

	ROBART 1	ROBART 2
line	1	7
box	0	9
circle	0	10
continuous line	1	7
continuous box	1	7
continuous circle	0	10
(cont. open box in ROBART 2)		

Other detailed tests, which asked for pairwise comparisons of shapes within each system and for individual rankings of difficulty also were generally in the direction predicted, but only some were statistically significant.

The overall pattern of results suggests that the design decisions made for selecting shapes in ROBART 2 led to action sequences that were easier to remember than those for ROBART 1, and, within ROBART 1, the sequences for some shapes were easier to remember than others, e.g., line was easier than continuous circle.

Structural Consistency

It is almost a truism that consistency of an interface will make it easier to use. What is less clear, however, is precisely what we mean by consistency and, more importantly, how to identify its absence. There are many kinds of consistency, for example: consistency of terminology (the same words (or actions) mean the same things in different contexts) and consistency of response (the same actions result in the same response). In this section we consider consistency of *structure*. With this notion we are trying to capture the idea that tasks or subtasks that are perceived as similar by the user are described by similar sequences of actions. For example, if the user perceives "drawing shapes" as one kind of task, then the sequence of steps used to draw one kind of shape should be similar to that used to draw any other kind. In this case we are dealing, not with individual, observable actions, but with sets of similar actions. We represent such sets of similar actions with nonterminal symbols in the grammar. Sequences of steps that are similar will thus have a similar underlying structure, and will derive from the same rules. Structural consistency is important, from the user point of view, because there will be fewer kinds of sequences to remember, and less chance of using one kind of sequence when another is correct.

In this section, we will present three examples, of increasing complexity, of structural (in)consistency.

Selecting Text Shapes: In ROBART 1, no action is required to select text, the keyboard is always available. This can be seen from rule 90, viz.,

90. select text shape ::= NULL

Since the length of NULL is, by convention, zero, this would appear at first to be the optimal design choice, since the string is as short as possible. However, there are other rules for selecting shapes which enter the picture. For example, to select an "1-c-b" shape (line, circle, or box) or a continuous shape the rules are

13. select 1-c-b shape ::= select 1-c-b switch + GO(1)
71. select new c-shape ::= select c-switch + GO(1)

Both require that switches be set.

Since these rules are of essentially the same form, they could be combined into one, more general rule, and later differentiated for the specific switch selections required. The more general rule would be (with the word "figural" standing for 1-c-b and continuous, combined):

rule a. select figural shape ::= select switch + GO(1)

There are thus at least two *necessary* rules for selecting shapes, in ROBART 1, rule a and rule 90.

ROBART 1 is thus "structurally inconsistent." Two different sequences of steps are required where one would suffice. We would expect the user to make errors because of this inconsistency. Having learned one "rule" for selecting shapes, not necessarily consciously, the user will apply it in circumstances which are not appropriate. This is equivalent to the child who has learned the rule, "verb + -ed makes past tense," and then insists on saying "yesterday I goed." Because of the structural inconsistency we would predict the following.

Prediction 4: A user who has learned to select other shapes will "expect" to select text in the same way, and will show this expectation by making "expectation responses."

Prediction 5: No equivalent responses will occur in ROBART 2.

While not formally tested, we did observe such "expectation responses." Users who had learned to select other shapes, when trying to select text for the first time, would extend their hand toward the switchbox, look over the switches as if looking for a text switch, or ask directly "which switch do I use."

In ROBART 2, on the other hand, one general rule suffices for all shapes and there is thus no expectation to unlearn, viz.,

rule. select shape ::= cursor in icon

We would therefore suggest that the ROBART 2 design decision was better than the one for ROBART 1. We do so on the assumption that erroneous actions or rules to unlearn are worse for the infrequent, naive user trying to learn or remember a system than slightly more physical action.

The required rule for ROBART 1 was quickly learned, however. The problems in the next example lasted longer.

Initiating Shapes: The user procedure for initiating a discrete shape, for example, a line, circle, or box, can be seen starting from rule 34:

34. initiate 1-c-b shape ::= cursor at start + start operation

That is, the user puts the cursor at some starting position, then presses the START and GO(2) buttons (rule 44). (The exact

knowledge required for starting lines, circles, and boxes differs and can be found from the appropriate rules.)

To initiate a continuous shape, however, rule 83 says

83. initiate c-shape ::= full know off + cursor at c-start + knob on

Thus, the user must turn the knob completely off to prevent unwanted "painting," move the cursor to the position where the painting is to start, then turn the knob on. This design decision was made to minimize hand action. A user, with his hand already on the knob, would simply keep it there.

We thus have two *necessary* rules in ROBART 1, one for discrete shapes and another for continuous ones. The two rules describe different sequences of actions for initiating the two sets of shapes.

In ROBART 2, one rule suffices for both discrete and continuous shapes:

initiate shape ::= cursor at start + EXECUTE

The user positions his cursor, then moves his hand to press the EXECUTE key.

We thus make the following predictions.

Prediction 6: In ROBART 1, users who have learned how to initiate discrete shapes will erroneously press the START and GO(2) buttons when attempting to initiate continuous shapes.

Prediction 7: No equivalent unnecessary action will occur in ROBART 2.

These predictions were in fact supported. In ROBART 1, 70 percent of the subjects erroneously pressed START and GO(2) at least once during the test, some of them many times. No corresponding erroneous action was observed in ROBART 2.

The ROBART 2 decision to have a consistent design (represented by fewer *necessary* rules) led to a system that was easier to remember, overall. We would therefore assert, for the user population in question, that this was a "better" design, even though more physical action was required than in ROBART 1.

Connecting Horizontal and Vertical Lines: Another, slightly more complicated example of structural inconsistency is the method for connecting horizontal and vertical lines in ROBART 1. Fig. 5 shows the structure for connecting ordinary lines and for connecting horizontal and vertical lines. The rules corresponding to these trees start with rule 55. Rules not necessary to the exposition have been omitted from the trees.

The language was designed for similarity at points marked 1 and 2. The rationale was the avoidance of extra work for the user. Once the first line has been drawn, successive endpoints, only, need to be indicated. However, in order for this to be possible, the language is inconsistent at points 3 and 4. No interrupt button is used following selection of horizontal line (since this would reset an index in the computer program which stores the latest endpoint). However, users who have learned that selecting shapes generally requires pressing the GO(1) button continue to do so at points 4 and 5. This is not

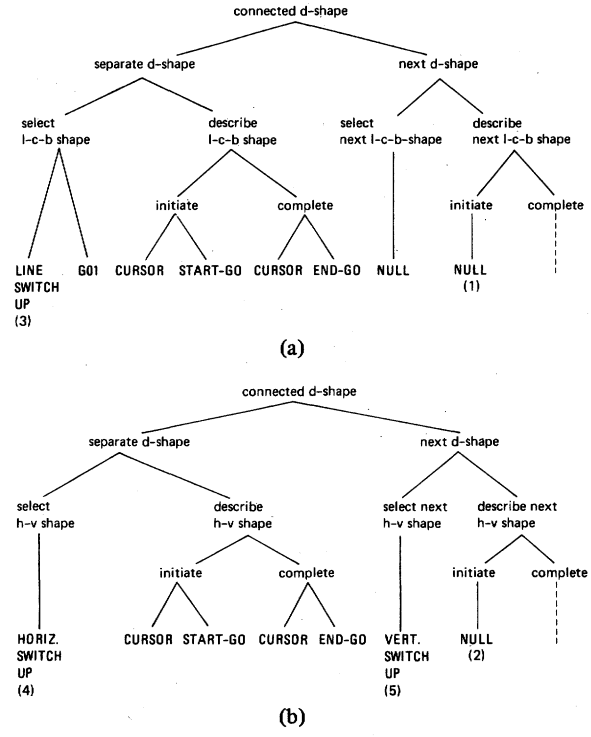

Fig. 5. Tree diagrams showing structure for (a) connected ordinary lines and (b) connected horizontal and vertical lines. Intermediate nodes have been omitted for simplicity.

catastrophic at point 4, merely an unnecessary action. But at point 5, the internal index is reset, the lines cannot be connected, and the user is confused. This again was informally observed, but not tested experimentally.

A better design would have been to require all lines to be initiated by a positive action, positioning the cursor and pressing START-GO(2). Again, slightly more work for the user would have been "better" than trying to be "efficient" with user actions, which caused confusion.

Discussion

Other Kinds of Terminal Symbols

In our formalism we have defined the terminal symbols as the actions a user would have to learn and remember ("cognitive terminal symbols"). Other kinds of terminal symbols, aimed toward design of other aspects of a system, are also possible. Thus to study the physical device in use for a particular action we would define "physical terminal symbols" (e.g., switchbox, screen). To describe where the user was looking we would define "visual terminal symbols," and to describe pure physical action we would define "action terminals" (for example, MOVE JOYSTICK, rather than (PUT) CURSOR AT DIAGONAL CORNER).

The physical and visual terminals could then be used to examine the assignment of function to hardware. One possible metric would be the number of alternations required for the terminal strings under study—the more *changes* in hand or eye position required, the more difficult the design. Change in hand or eye position is a recurring theme in human factors. Using the number of such changes as a possible metric pro-

vides a method for judging the icon versus switch distinction in the two ROBART versions.

Action terminal symbols would be of primary interest in describing systems for users doing repetitive tasks, while cognitive terminal symbols are more important for the nonroutine task and the naive user.

Assumptions

We have made a number of assumptions in discussing design decisions which should be opened up to discussion. There are at least three characteristics of a language which influence its ease of use: the number of terminal symbols, the lengths of the terminal strings, and the number of (forms of) rules. Common practice, based on sound engineering practice, is to minimize the first (e.g., the number of switches), occasionally the second, and rarely the third. We would assert that, all else being equal, the order of importance for naive users doing nonroutine tasks is just the reverse.

Our reason for this assertion is that human memory limitation is a major problem in systems for naive or infrequent users, and rules describe more of system than do strings, and strings more than terminal symbols. (Rules describe sets of sequences of actions, strings describe single sequences of actions, and terminals describe single actions.)

Rules and Predictability

We have suggested that learning an action language requires learning *rules*, at least unconsciously. The intuitive notion that a system should be "consistent" can be made somewhat more precise by the notion of *necessary* rules that we introduced informally above.

The consistency of some aspects of a design can be tested with pencil and paper, before a system has been built, by use of *prediction tests*. These would require teaching a subject the actions corresponding to some rule, then asking him to guess at another. The more *predictable* a design, the easier it should be to learn and remember. In fact, predictability of a design, suitably quantified, should be one measure of the "goodness" of a design.

Overall Results of Tests

While a study of overall learning time and accuracy was not our major focus, the results are of interest in showing that, for the same function and differing interface designs, measurable differences in user performance can be found.

On the average, subjects spent less time learning ROBART 2 than ROBART 1 (51 min and 76 min, respectively), roughly a 50 percent difference. These are not total times required to learn the systems by self-teaching, since a maximum time was set, some subjects did not complete the entire learning process and some required experimenter help. Since the methods were consistent, however, the comparisons are meaningful.

Subjects also made fewer errors in ROBART 2 than ROBART 1 (an average of 6.4 and 8.6, respectively, out of a maximum of 57). The results did not arise from a few deviant subjects, but were true for most of them. Nine of the ten subjects learned ROBART 2 faster than ROBART 1, no matter which system was learned first, and nine of the ten also made fewer errors with ROBART 2 than ROBART 1. (Nine out of ten is statistically significant with a sign test, $p < 0.05$, one-tailed.) The single subject in each case who did not do better with ROBART 2 learned ROBART 2 first, and one would, of course, expect the first system learned to be harder.

All subjects said, when questioned, that they found the "SCREEN ROBART" easier to learn and remember than the "BOX ROBART," although one surprisingly preferred the latter. (She spent more time learning ROBART 1, thus felt she knew it better, and she also preferred the switches because she felt "more in control.")

Other Related Work

The notion that user actions at a terminal can be viewed as a language has been clearly expressed in Foley and Wallace [4] and Foley [5]. An early suggestion that BNF description might be used to predict *psychological* difficulty of user languages (query languages) can be found in [13]. A short but provocative note by Ledgard and Singer [7] reports that several different formal notations were used before coding an interactive editor and claims that formal definition should be used during the design process.[1] In addition, we have been able to locate two instances of somewhat related work.

Embley [3] studied one particular control construct for programming interactive dialogues for computer-aided instruction (the KAIL selector). His objective was to apply both empirical and formal methods to assist in the design of this new control construct. He first compared the proposed KAIL selector, experimentally, with an ALGOL-like construct, using a CAI system to measure "understandability" of the two constructs. (At this stage, design of the KAIL selector was based on observation of factors such as levels of nesting and length of code rather than on formal analysis.) He then examined the semantics (not syntax) of variations of the KAIL selector itself, using a formal notation, to help select between them. The variants were examined formally, not experimentally.

While this work concentrates on one particular construct rather than an entire language, and is concerned with a traditional programming language rather than an action language, it is important in that it exemplifies the use of both empirical and formal methods to aid in language design.

Moran [9] presents a formalism he has devised, CLG (command language grammar), to show the "general structure" of command language systems. He attempts to describe all "levels" of a system, from the conceptual to the physical device level. His formalism is an English-like notation with predefined primitives, e.g., (WHEN IS, THE PROMPT FOR, THE ACTION FOR, THE RESPONSE TO), a programming-like construction (DO loops), and production rules. He illustrates his concepts with a simple artificial system for managing message files, based on a single machine-prompt-followed-by-user-response

[1] This report was brought to our attention during the review process. It is unfortunately brief (5 pages), with little detail and no experimental testing. However, the work is exciting because it reports a major attempt to use formal methods in design and presents the impressions of well-known researchers on the value of this process.

interaction. He describes a complete user-system interaction (rather than concentrating on an action language—a sequence of user actions).

Moran modestly claims that his CLG is "just a framework," with extensions to help in the design process planned for the future. While his example is an artificial one (thus precluding behavioral tests) describing a single one prompt-one response type system, the work is important in its attempt to fully describe all levels of a system using formal methods.

Beyond Simple BNF

We originally chose a BNF-like notation to describe the ROBART action languages because it is commonly used for describing other kinds of languages and has several properties that we liked. (It is relatively compact, is easy to manipulate automatically, and can describe an infinite language with a finite set of rules.) It soon became apparent, however, that a straightforward BNF-like notation was not the ultimate choice. We wanted a notation that would 1) describe all and only the legal strings for the user and 2) show the structure of the language with as few nearly redundant rules as possible.

These turned out to be somewhat contradictory requirements unless we introduced the stratagem of "semantic restrictions" to prevent the generation of illegal strings. Without the semantic restriction given in the footnote for rule 9, for example, selecting box, then describing circle would be a permissible string.

We could describe all and only the legal strings by a very lengthy grammar which listed each shape and by then redundantly having a rule for selecting and describing each one. But such a description would lose the general structure—that most shapes are constructed in similar ways. On the other hand, we could write a more terse grammar than the one we have presented, showing more of the general structure, but at the expense of more semantic restrictions. For example, if we had one general rule saying that any shape had to be selected and described (rather than rule 9 for discrete shapes, and rule 68 for continuous ones), we would then need a semantic restriction to prevent selecting a discrete shape and describing a continuous one.

A simple, straightforward BNF notation is clearly usable now. We can write two different forms of grammars, a lengthy "legal string grammar"—of possible use in providing feedback to users about incorrect action sequence—and a "structure revealing grammar" to locate design inconsistencies.

However, in the long run better notational schemes need to be found or devised which reveal both structure and legal strings. While some pattern revealing formalism is required, and while BNF is an obvious first choice to explore and is clearly useful, it may not be the final solution.

The Value of Formal Description

Analytical Tool: A major value of a formal description of an action language is that of any analytic tool: we prefer to analyze a paper and pencil representation of a system rather than waiting until we have a working model. Formal analysis will not preclude all behavioral testing, but it may give early warning of some design flaws and simplify making some design choices.

Precision: Another value of a formalism is simply that it forces us to be precise (and concise). The simple act of describing a system, even without comparing alternatives, can lead to interesting insights. For example, only when writing a formal description of ROBART 1 did the author realize that switches had to be "undone," in spite of many reminders to users to "turn off the other switches."

Testable Hypotheses: The precision of a formal description also helps in formulating clear, testable hypotheses about design decisions. We could quite clearly specify, for example, that we were testing "initiating actions," "selection actions," etc.

Quantifiable, General, Intrinsic Properties of Easy-to-Use Systems: The formalism also permits looking for properties of a language such as length of string or number of necessary rules, which will apply generally to other action languages. (A paper by Wang [16] suggests other measures of string complexity in English, some of which may be applicable here.) We would prefer to be able, ultimately, to identify intrinsic characteristics of a language which make it easy to learn and use, rather than rely exclusively on behavioral testing. Furthermore, we would like to quantify such characteristics and show their behavioral correlates. And we would like automatic methods for locating design flaws. (We used a quasi-automatable technique.)

Automatic Manipulation: A formal notation can, of course, be automatically manipulated. Thus, it should be possible to construct and examine any desired combination of strings (to construct testing materials of known relative difficulty, for example).

Explanation of User Errors: User errors can sometimes be explained on the basis of a formalism. While testing provides data, it does not necessarily provide explanation of the data obtained. For example, in ROBART 2, we observed that users unnecessarily pressed EXECUTE after selecting colors. It was clear on the basis of the formalism that the structure as designed was different from the structure inferred by users, i.e., the designer's model and the user's model disagreed.

In sum, any technology requires analytic tools, testable hypotheses based on theoretical constructs, and a body of knowledge built on testing such constructs. Formal notation appears to have potential as one such tool.

SUMMARY

The field of human factors is frequently judged to be lacking in rigor. To counteract this judgment, we have shown, by means of a concrete example, that 1) user actions at a terminal can be described by means of a formal grammatical notation, 2) the formalism can be used to make predictions for comparing design alternatives and locating design flaws, and 3) the predictions can be empirically tested.

We have given a complete formal description of the ROBART 1 system and then made predictions on the basis of the formal description of ROBART 1 and matching portions of ROBART 2. We predicted that selecting shapes in RO-

BART 1 would vary in difficulty, that selecting shapes in ROBART 2 would *not* vary in difficulty, and that selecting any ROBART 2 shape would be easier than selecting the same shape in ROBART 1. These predictions were upheld in objective testing of immediate memory and in subjective questionnaires. We also predicted that users would make errors in attempting to initiate continuous shapes using ROBART 1 and would not make mistakes in ROBART 2. These predictions were also supported in testing.

Formal description of interactive systems is an analytic tool. It makes possible the examination of a design from the human point of view early in the design cycle. We can thus compare some aspects of alternative designs and identify some design inconsistencies before a working model is built.

The method we have presented is a general one. Other interactive systems (word processors, editors, etc.) can be described by means of a formal notation. The predictions we made were based on general properties of the formal description and the approach should therefore generalize.

Appendix

ROBART 1 Grammar

In the following description, terminal symbols are indicated in small capital letters, nonterminals in lowercase. The rules are numbered for convenience of reference. No ordering is implied. Headings have been inserted to facilitate locating sections of the grammar. The resulting sections are to be used as a rough guide only.

Picture

1. picture ::= colored shape | picture + colored shape

Colors

2. colored shape ::= color + shape | shape + color
3. color ::= new color | old color | starting default color
4. new color ::= CURSOR IN RED | CURSOR IN BLUE | CURSOR IN GREEN | . . .
5. old color ::= NULL
6. starting default color ::= NULL

Shapes

7. shape ::= discrete shape | continuous shape | text shape
8. discrete shape ::= separate d-shape | connected d-shape

Separate Discrete Shapes

9. separate d-shape ::= select separate d-shape + describe separate d-shape[2]
10. select separate d-shape ::= select old d-shape | select new d-shape
11. select old d-shape ::= NULL
12. select new d-shape ::= select 1-c-b shape | select h-v shape

Switches for Discrete Shapes

13. select 1-c-b shape ::= select 1-c-b switch + GO(1)
14. select h-v shape ::= select h-v switch
15. select 1-c-b switch ::= select line | select box | select circle
16. select h-v switch ::= select horizontal | select vertical
17. select line ::= undo non-line + set line
18. undo non-line ::= BOX SWITCH DOWN | CONTINUOUS SWITCH DOWN | HORIZ SWITCH DOWN | VERT SWITCH DOWN | . . .[3] (all ordered combinations of above)
19. set line ::= LINE SWITCH UP
20. select box ::= undo non-box + set box
21. undo non-box ::= LINE SWITCH DOWN | CONTINUOUS SWITCH DOWN | HORIZ SWITCH DOWN | VERT SWITCH DOWN | . . .[3] (all ordered combinations of above)
22. set box ::= BOX SWITCH UP
23. select circle ::= undo non-circle + set circle
24. undo non-circle ::= CONTINUOUS SWITCH DOWN | VERT SWITCH DOWN | HORIZ SWITCH DOWN | . . .[3] (all ordered combinations of above)
25. set circle ::= set line + set box | set box + set line
26. select horizontal ::= undo non-horizontal + set horizontal
27. undo non-horizontal ::= LINE SWITCH DOWN | BOX SWITCH DOWN | VERT SWITCH DOWN | CONTINUOUS SWITCH DOWN | . . .[3] (all ordered combinations of above)
28. set horizontal ::= HORIZ SWITCH UP
29. select vertical ::= undo non-vertical + set vertical
30. undo non-vert ::= LINE SWITCH DOWN | BOX SWITCH DOWN | HORIZ SWITCH DOWN | CONTINUOUS SWITCH DOWN | . . .[3] (all ordered combinations of above)
31. set vertical ::= VERT SWITCH UP

Describing (Indicating Position, Location, etc.) of Discrete Shapes

32. describe separate d-shape ::= describe 1-c-b shape | describe h-v shape
33. describe 1-c-b shape ::= initiate 1-c-b shape + complete 1-c-b shape[4]
34. initiate 1-c-b shape ::= cursor at start + start operation
35. cursor at start ::= line start | circle start | box start
36. line start ::= cursor at line point(1)
37. circle start ::= cursor at circle center

[2]The shapes selected and described must be the same. For example, if "line" is selected, then "line" must be described.

[3]The shapes undone must be the ones previously set.

[4]The shapes initiated and completed must be the same.

38. box start ::= cursor at box corner
39. complete 1-c-b shape ::= cursor at end + end operation
40. cursor at end ::= line end | circle end | box end
41. line end ::= CURSOR AT LINE POINT(2)
42. circle end ::= CURSOR AT CIRCUMFERENCE
43. box end ::= CURSOR AT DIAGONAL CORNER
44. start operation ::= START-GO(2)
45. end operation :: END-GO(2)
46. describe h-v shape ::= initiate h-v shape + complete h-v shape[4]
47. initiate h-v shape ::= cursor at h-v start + start operation
48. cursor at h-v start ::= horiz start | vert start
49. horiz start ::= CURSOR AT H-LINE POINT(1)
50. vert start ::= CURSOR AT V-LINE POINT(1)
51. complete h-v shape ::= cursor at h-v end + end operation
52. cursor at h-v end ::= horiz end | vert end
53. horiz end ::= CURSOR AT H-LINE END POINT ON Y-AXIS
54. vert end ::= CURSOR AT V-LINE END POINT ON X-AXIS

Connected Shapes

55. connected d-shape ::= separate d-shape + series d-shape
56. series d-shape ::= next d-shape | next d-shape + series d-shape
57. next d-shape ::= next 1-c-b shape | next h-v shape
58. next 1-c-b shape ::= select next 1-c-b shape + describe next 1-c-b shape[2]
59. select next 1-c-b shape ::= NULL
60. describe next 1-c-b shape ::= initiate next 1-c-b shape + complete next 1-c-b shape[4]
61. initiate next 1-c-b shape ::= NULL
62. complete next 1-c-b shape ::= complete 1-c-b shape
63. next h-v shape ::= select next h-v shape + describe next h-v shape[2]
64. select next h-v shape ::= select h-v switch (alternate)
65. describe next h-v shape ::= initiate next h-v shape + complete next h-v shape[4]
66. initiate next h-v shape ::= NULL
67. complete next h-v shape ::= complete h-v shape

Continuous Shapes

68. continuous shape ::= select c-shape + describe c-shape[5]
69. select c-shape ::= select old c-shape | select new c-shape
70. select old c-shape ::= NULL
71. select new c-shape ::= select c-switch + GO(1)
72. select c-switch ::= CONTINUOUS SWITCH UP + select 1-c-b switch | select 1-c-b switch + CONTINUOUS SWITCH UP

Describing Continuous Shapes

73. describe c-shape ::= set knob + initiate c-shape + continue c-shape + complete c-shape
74. set knob ::= fix width-vary angle | fix angle-vary width
75. fix width-vary angle ::= WIDTH-ANGLE SWITCH DOWN + knob on + WIDTH-ANGLE SWITCH UP
76. fix angle-vary width ::= WIDTH-ANGLE SWITCH UP + knob on + WIDTH-ANGLE SWITCH DOWN
77. knob on ::= full knob on | partial knob on
78. full knob on ::= ROTATE KNOB FULL CLOCKWISE
79. partial knob on ::= ROTATE KNOB PARTIAL CLOCKWISE
80. knob off ::= full knob off | partial knob off
81. full knob off ::= ROTATE KNOB FULL COUNTERCLOCKWISE
82. partial knob off ::= ROTATE KNOB PARTIAL COUNTERCLOCKWISE
83. initiate c-shape ::= full knob off + cursor at c-start + knob on
84. cursor at c-start ::= POSITION CURSOR
85. continue c-shape ::= change knob + change cursor position
86. change knob ::= knob on | partial knob off | NULL
87. change cursor position ::= MOVE CURSOR | NULL
88. complete c-shape ::= full knob off

Text Shapes

89. text shape ::= select text shape + describe text shape
90. select text shape ::= NULL
91. describe text shape ::= select text background + select text size + describe text typing | select text size + select text background + describe text typing
92. select text background ::= ADDITIVE-BLOCKED SWITCH UP | ADDITIVE-BLOCKED SWITCH DOWN
93. select text size ::= SINGLE-DOUBLE SWITCH UP | SINGLE-DOUBLE SWITCH DOWN
94. describe text typing ::= initiate typing + continue typing + complete typing
95. initiate typing ::= POSITION CURSOR + start operation
96. continue typing ::= typing action | continue typing + typing action
97. typing action ::= symbol operation | typing control operation
98. symbol operation ::= symbol | symbol operation + symbol
99. symbol ::= A | B | . . .Z | 1 | 2 | . . .9 | 0 | * | ? | . . .
100. typing control operation ::= SHIFT | | | | | | | . . .
101. complete typing ::= NULL

Acknowledgment

The author wishes to thank the following for their careful reading and comments on this paper (and/or an earlier version): J. L. Bennett, E. D. Carlson, R. Strong, J. A. Sutton, D. L. Weller, and R. Williams. She also wishes to acknowledge the following for many interesting discussions: L. Barbosa, B. C. Housel, F. P. Palermo, and S. N. Zilles. She also wishes to acknowledge the following for their part in design and/or development in ROBART 1 or ROBART 2: M. Breternitz (ITA, Sao Paulo, Brazil), A. Fan (Mills College), G. M. Giddings, G. G. Langdon, F. P. Palermo, D. Raimondi, R. M. Revelle, D. Silberberg (MIT), D. L. Weller, and R. Williams.

References

[1] E. D. Carlson, G. M. Giddings, and R. Williams, "Multiple colors and image mixing in graphics terminals," in *Inform. Processing 77, IFIP*. North-Holland, 1977, pp. 179–182.

[2] N. Chomsky, *Syntactic Structures*. The Hague, The Netherlands: Mouton, 1964.

[3] D. W. Embley, "Empirical and formal language design applied to a unified control construct for interactive computing," *Int. J. Man-Mach. Studies*, vol. 10, pp. 197–216, Mar. 1978.

[4] J. D. Foley and V. L. Wallace, "The art of natural graphic man-machine conversation," *Proc. IEEE*, vol. 62, pp. 462–471, Apr. 1974.

[5] J. D. Foley, "The structure of interactive command languages," in *Methodology of Interaction*, R. A. Guedj *et al.*, Eds. North-Holland: 1980, pp. 227–234.

[6] G. G. Langdon, Jr., P. Reisner, and D. Silberberg, "Robart 2: A stand-alone graphics terminal system for color slides," IBM Res. Rep. RJ2871, San Jose, CA, July 1980.

[7] H. F. Ledgard and A. Singer, "Formal definition and design," COINS Tech. Rep. 78-01, Univ. Massachusetts, Amherst, Feb. 1978.

[8] P. M. Lewis, II, D. J. Rosenkranz, and R. E. Stearns, *Compiler Design Theory*. Reading, MA: Addison-Wesley, 1976.

[9] T. P. Moran, "The command language grammar," *Int. J. Man-Mach. Studies*, vol. 14, to be published.

[10] R. A. Morrison, "Graphic language translation with a language independent processor," in *AFIPS Conf. Proc., Fall Joint Comput. Conf.*, vol. 31. Washington, DC: Thompson, 1967, pp. 723–731.

[11] J. F. Musgrave, "Experiments in computer-aided graphic expression," *IBM Syst. J.*, vol. 17, no. 3, pp. 241–259, 1978.

[12] P. Naur, Ed., "Revised report on the algorithmic language ALGOL," *Commun. Ass. Comput. Mach.*, vol. 6, pp. 1–17, Jan. 1963.

[13] P. Reisner, "Use of psychological experimentation as an aid to development of a query language," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 218–229, May 1977.

[14] R. G. Schoup, "Towards a unified approach to 2-D picture manipulation," *ACM SIGGRAPH Comput. Graphics*, vol. 11, p. 178, Summer 1977.

[15] S. Siegel, *Non-Parametric Statistics*. New York: McGraw-Hill, 1956.

[16] M. D. Wang, "The rule of syntactic complexity as a determiner of comprehensibility," *J. Verbal Learning and Verbal Behavior*, vol. 9, pp. 398–404, Aug. 1970.

Phyllis Reisner received the A.B. degree in English from Hunter College, New York, NY, in 1955, and the M.S. and Ph.D. degrees in information science from Lehigh University, Bethlehem, PA, in 1971 and 1972, respectively.

After graduating from Hunter College, she studied at The Sorbonne, Paris, France, under a French Government grant and a Fulbright Travel grant. In 1960, she joined the IBM Research Division, and is currently at the IBM Research Laboratory, San Jose, CA. Her research interests focus on human factors of man-machine systems, particularly development of techniques for improving design of user languages.

Dr. Reisner is a member of the Association for Computing Machinery, the American Psychological Association, and the Human Factors Society.